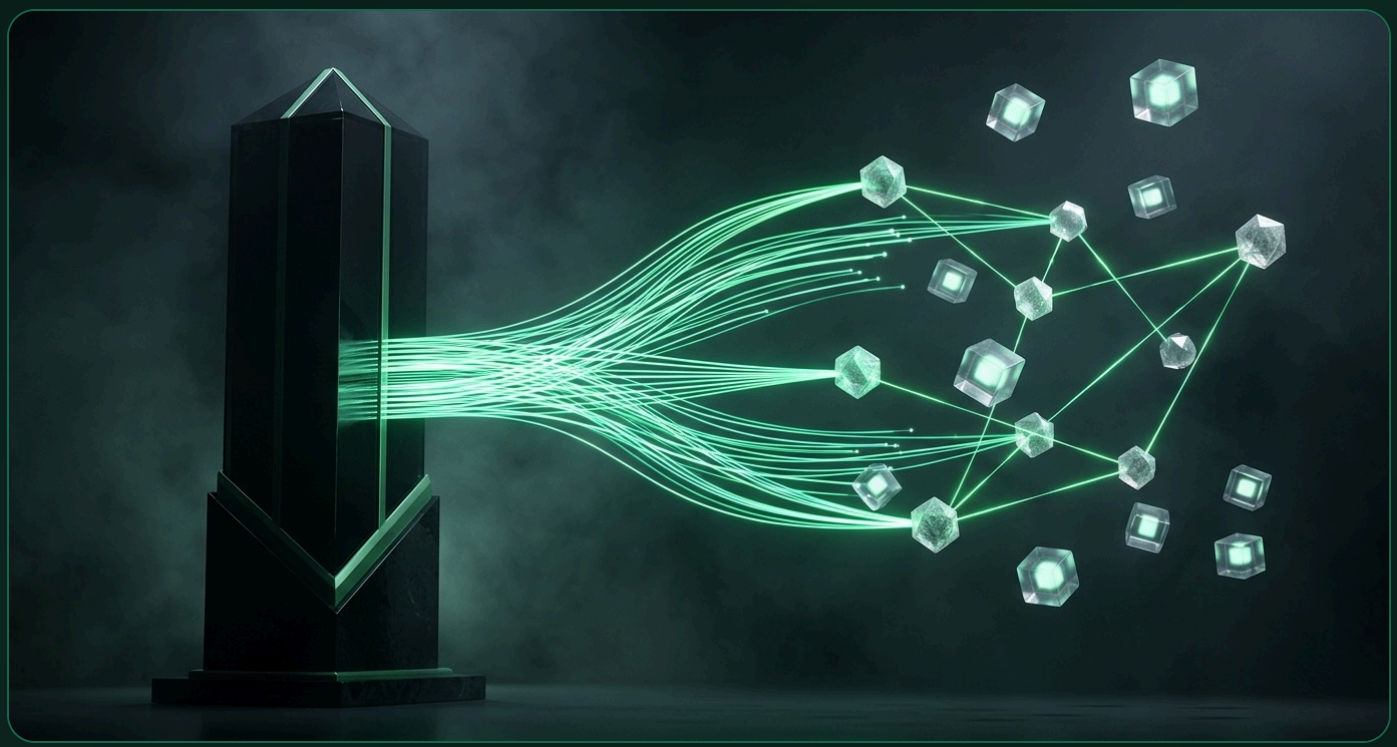
OPENAI CODEX, AGENTS & REALTIME STUDIOS • FIELD GUIDE

Codex CLI & Codex Cloud

Mastering OpenAI's twin engineering engines—combining millisecond-fast local terminal execution on `GPT-6.1 Sol` with isolated, asynchronous Codex Cloud worker fleets that turn backlogs into review-ready pull requests.

`~/.codex/config.toml` `AGENTS.md` `SKILL.md` Modules GPT-6.1 Sol 3 Sandbox Tiers Cloud PR Fleet



Sandboxed Terminal Speed

Run rapid edit-test loops locally on `GPT-6.1 Sol` with deterministic `read-only`, `workspace-write`, and `danger-full-access` isolation.

Hierarchical Repo Memory

Encode team rules once in `AGENTS.md` and keep base prompts lean via progressive `SKILL.md` modules.

Parallel Cloud PR Fleet

Hand off heavy migrations and overnight issue queues via `/cloud` into isolated containers that ship tested PRs.

Local Terminal Speed Meets Isolated Cloud Fleet

Modern software engineering splits naturally into two rhythms: tight, interactive problem solving at your keyboard and heavy, parallelizable chores you would rather delegate overnight. Codex bridges both with one unified configuration, rulebook, and skill format.

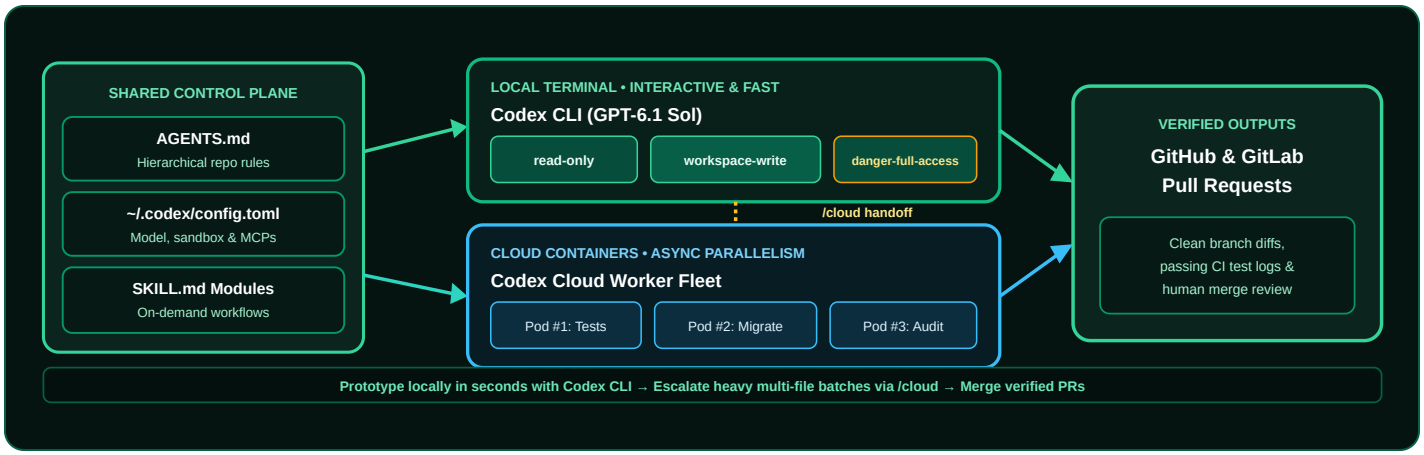
Quick Navigation

1. Hybrid Architecture & Mental Model p. 2
2. Setup, GPT-6.1 Sol & Sandbox Matrix p. 3
3. Annotated Codex Terminal UI Tour p. 4
4. Mastering `config.toml` & `AGENTS.md` p. 5
5. Modular `SKILL.md` & Prompt Anatomy p. 6
6. Troubleshooting Clinic & Pre-Flight p. 12
7. Terminal Power & TDD Fix Loops p. 7
8. Codex Cloud Fleets & PR Review Loops p. 8
9. 5 Real-World Copy-Paste Recipes p. 9
10. Security, Secrets & Egress Governance p. 10
11. Cross-Tool Synergy Pipeline p. 11
Golden Rule: Narrowest Sandbox First All

One Brain, Two Execution Surfaces

Both **Codex CLI** (running interactively in your shell) and **Codex Cloud** (running asynchronously in ephemeral cloud sandboxes) read the exact same repository constitution (`AGENTS.md`), modular playbooks (`SKILL.md`), and runtime preferences (`~/.codex/config.toml`). You never maintain two sets of instructions: prototype an approach locally in seconds, then spin up a fleet of parallel cloud workers when the task scales across dozens of files or branches.

Unified Architecture: Shared Rules Feeding Local & Cloud Runtimes



Mental Model Takeaway: Same Constitution, Different Blast Radius

Think of Codex CLI as your pair programmer sitting right inside your working tree, and Codex Cloud as a remote engineering pod cloning your repo into a clean container. Because both enforce `AGENTS.md` and `SKILL.md`, habits you teach locally scale automatically across your entire cloud fleet.

Setup, `GPT-6.1 Sol` & Approval-Sandbox Matrix

Getting started with Codex CLI takes under sixty seconds. Install the binary, authenticate with your ChatGPT or OpenAI API account, and choose the narrowest sandbox and approval pairing your current task requires.

1 60-Second Install & Login

```
# Install globally via npm or Homebrew
npm install -g @openai/codex
brew install --cask codex

# Sign in via OAuth (ChatGPT Plus/Pro/Team) or API key
codex login
codex --version
```

Subscription sign-in shares workspace quotas; API key billing scales seamlessly for high-concurrency CI and Cloud batches.

2 `GPT-6.1 / 6.2 Sol` & `Astra` Tiers

- **500k Default (1.05M Max):** `gpt-6.1-sol` ($2/$10 per 1M API tokens; $0.10 cached) & **GPT-6.2 Sol/Astra** (Nov 17, 2026) hold full modules; Free/Go desktop uses `gpt-6-luna`.
- **High-Velocity Reasoning:** Tuned specifically for terminal tool calls, patch synthesis, and fast edit-compile-test feedback loops.
- **Adaptive Effort & Worktrees:** Switch models via `/model`, manage parallel Git worktrees natively in CLI (`0.171+`), or use the Codex view inside the unified **ChatGPT Desktop app**.

Decision Matrix: Approval Policies & Sandbox Isolation Tiers

Axis / Policy	`read-only` Sandbox	`workspace-write` Sandbox	`danger-full-access`
What Codex Can Touch	Reads repository files and inspects git status; **all file writes and mutating shell calls blocked**.	Reads and writes inside your project directory under OS sandboxing (Seatbelt/Landlock); **network off by default**.	Disables OS sandboxing completely, granting direct host filesystem and outbound network access.
`on-request` **Policy** Default & Safest	Runs read-only inspections smoothly; pauses and asks before stepping outside safe bounds.	**Sweet Spot:** Runs edits and local tests cleanly; prompts if leaving workspace or accessing network.	Prompts before escalating outside trusted commands; catches destructive calls on host machines.
`never` **Policy** Non-Interactive	Executes analysis without prompting; any attempted disk write fails cleanly against the sandbox.	Runs unattended inside the workspace, or pair with `--approve-for-me` (`auto_review`) for AI review.	Zero prompts and zero isolation (`--yolo`); reserve strictly for throwaway VMs or CI pods.
`granular` **Policy** Fine-Grained	Uses TOML booleans (`sandbox_approval`, `rules`, `mcp_elicitations`) to gate tool categories.	Auto-permits safe repo edits while prompting specifically for MCP elicitations or skill scripts.	Enforces explicit human confirmation gates on sensitive host mutations and external connectors.

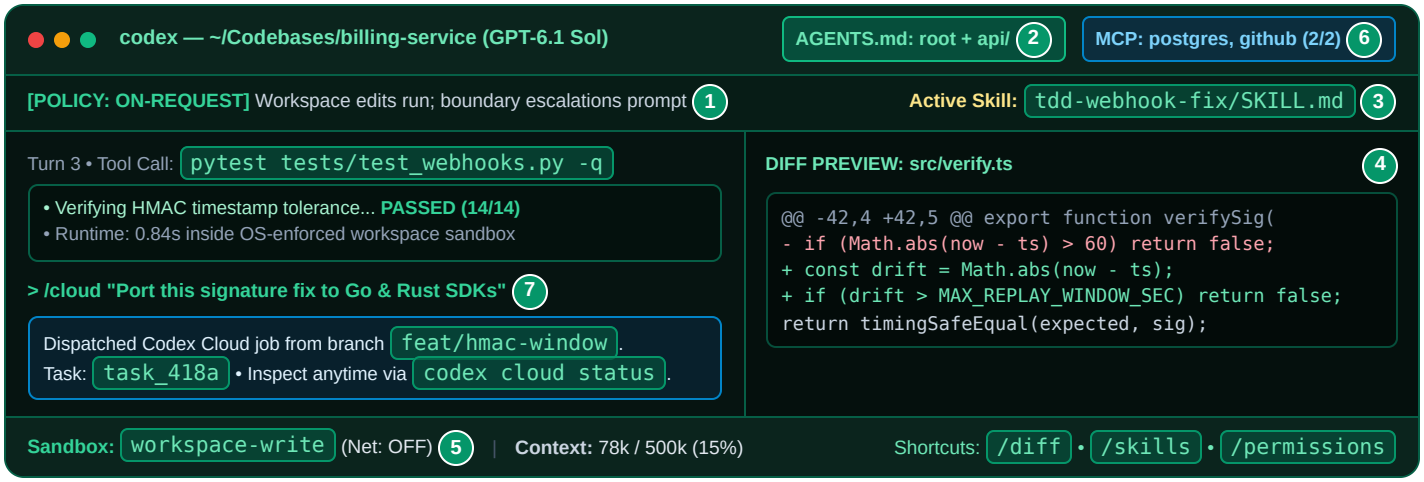
```
# Daily driver launch: workspace-write sandbox with on-request approval prompts
codex --sandbox workspace-write --ask-for-approval on-request "Fix failing webhook signature tests"

# Safe security audit: zero disk writes allowed
codex --sandbox read-only --ask-for-approval on-request "Audit auth middleware for IDOR risks"
```

Golden Pairing Rule: Start in `workspace-write` + `on-request`

Never jump straight to `danger-full-access` out of convenience. Running `on-request` inside `workspace-write` lets `GPT-6.1 Sol` iterate through compile-and-test cycles smoothly while guaranteeing it cannot touch `~/.ssh`, sibling repositories, or live production APIs without asking.

Annotated Tour: Codex CLI Terminal Cockpit

Codex CLI packs policy awareness, live diffs, sandbox boundaries, and cloud handoff into one keyboard-first terminal cockpit. Study these seven touchpoints before running your first autonomous loop.



Interface Legend: The 7 Live Codex CLI Touchpoints

1 **Approval Policy Banner:** Displays active policy (`on-request` , `never` , or `granular` / `auto_review`) so you know when actions pause for sign-off.

2 **Loaded `AGENTS.md` Badge:** Confirms every active instruction file merged from global config, repo root, and the current working directory.

3 **Active `SKILL.md` Indicator:** Lights up when Codex loads a specialized modular skill on demand instead of guessing ad-hoc steps.

4 **Inline Diff Hunk Preview:** Shows exact line-by-line edits and test results before you stage commits or invoke `/diff` for full review.

5 **Sandbox & Network Meter:** Verifies your active isolation boundary (`read-only` , `workspace-write` , or `danger-full-access`) and whether network egress is sealed.

6 **MCP Server Status Pill:** Confirms configured tools from `~/.codex/config.toml` (such as Postgres or GitHub) connected cleanly at startup.

7 **Seamless `/cloud` Handoff Command:** Escalates the current branch, context summary, and parallel task fan-out straight from your local terminal into asynchronous Codex Cloud worker containers without re-explaining the problem.

Mastering `config.toml`, MCPs & `AGENTS.md`

Two plain-text files govern almost everything Codex does: `~/.codex/config.toml` controls your model, sandbox defaults, and MCP tools, while hierarchical `AGENTS.md` files serve as your repository's living engineering constitution.

1. `~/.codex/config.toml` — Defaults & MCP Wiring

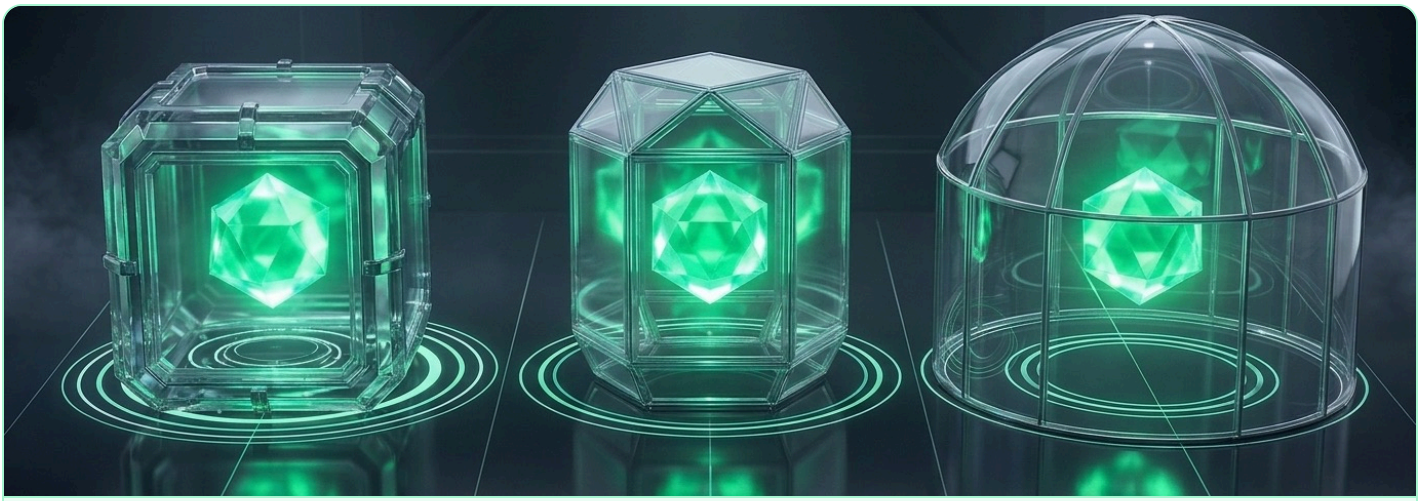
```
model = "gpt-6.1-sol"
approval_policy = "on-request"
sandbox_mode = "workspace-write"

[sandbox_workspace_write]
network_access = false

[mcp_servers.postgres_ro]
command = "npx"
args = ["-y", "@modelcontextprotocol/server-postgres"]
env = { DATABASE_URL = "${STAGING_RO_URL}" }

[mcp_servers.docs]
url = "https://mcp.internal.company.dev/sse"
```

2. Hierarchical `AGENTS.md` Cascade

1. **Global (`~/.codex/AGENTS.md`):** Personal preferences—preferred test flags, commit style, never touch `.env`.
2. **Repo Root (`./AGENTS.md`):** Architecture map, build/lint commands, and PR expectations shared via Git.
3. **Subdirectory (`./services/billing/AGENTS.md`):** Local overrides loaded only when editing files inside that service folder.

Keep `AGENTS.md` Actionable and Under 150 Lines

Treat `AGENTS.md` as an executive briefing, not a novel: list exact test commands (`pnpm test --filter api`), forbidden directories, and architectural boundaries. Push rare or step-heavy procedures into on-demand `SKILL.md` modules instead.



Defense in Depth Across Three Isolation Domes: Whether Codex runs inside a sealed `read-only` vault, a jailed `workspace-write` prism, or a `danger-full-access` workshop, `config.toml` enforces tool boundaries deterministically while `AGENTS.md` supplies project judgment.

On-Demand Skills: Keep Context Lean with `SKILL.md`

Stuffing twenty playbooks into `AGENTS.md` wastes precious context tokens and distracts the model. Codex solves this with **progressive disclosure**: only tiny skill names and trigger descriptions load at startup, while full `SKILL.md` instructions load only when a task matches.

Skill Folder Layout (User or Repo Scoped)

```
~/.agents/skills/db-migration/   # Personal across all
repos
# or <repo>/.agents/skills/db-migration/
├── SKILL.md        # YAML frontmatter + step-by-step
playbook
├── scripts/        # Optional deterministic verification
scripts
└── references/     # Loaded only when deep schema docs
needed
```

Frontmatter Index (Always Loaded, ~30 Tokens)

```
---
name: db-migration
description: Create, verify, and test reversible SQL
  migrations in packages/db. Use whenever adding columns,
  indexes, or altering Prisma/Postgres schemas.
---
```

Anatomy of a High-Precision `SKILL.md` Specification

Great skills trigger reliably without false positives. Combine these five color-coded building blocks inside your frontmatter and instructions:

[1. Crisp Trigger Description] [2. File Scope] [3. Ordered Workflow] [4. Scripts & MCPs] [5. Stop & Safety Guard]

Trigger when the user requests schema changes, table additions, or migration rollbacks. Restrict edits strictly to packages/db/migrations/ and packages/db/schema.sql. Step 1: Inspect existing migration order. Step 2: Author forward + down SQL. Step 3: Run reset test twice to prove idempotency. Execute scripts/check-locks.sh and inspect staging schema via [mcp_servers.postgres_ro]. Never drop non-empty columns without an explicit two-phase deprecation note; halt and report if drift is detected.

3-Stage Progressive Loading Lifecycle

Loading Stage	What Enters Context	Why It Boosts `GPT-6.1 Sol` Accuracy
Stage 1: Discovery	`name` + `description`	Costs only ~25–40 tokens per skill so you can keep 30+ specialized workflows installed with near-zero prompt bloat.
Stage 2: Activation	Full `SKILL.md` body	Injected only after your prompt matches the trigger description (or when you invoke `$skill-name` explicitly).
Stage 3: Deep Assets	`scripts/` & `references/`	Read selectively during execution; deterministic helper scripts execute directly without pasting huge boilerplate into chat.

Pro Tip: Write Trigger Descriptions Around User Intent, Not Implementation

Avoid vague descriptions like “Database helper.” State concrete situations (“Use when modifying SQL tables, indexes, or fixing migration checksum errors”) so Codex activates the right skill automatically on the first turn.

Terminal Power Workflows with `GPT-6.1 Sol`

Because `GPT-6.1 Sol` pairs a 500k default (1.05M max) token window with low-latency tool execution, Codex CLI shines at tight verify-as-you-go loops. Give it a concrete verification signal up front and watch it self-correct before you review a single line.

1 Test-Driven Fix Loops (Red → Green)

Never ask Codex to “fix the bug” in the abstract. Point it at a failing test or tell it to reproduce the failure first:

- **Reproduce:** Write a failing unit test capturing the edge case and confirm it fails.
- **Patch Narrowly:** Edit only the responsible module while keeping public types intact.
- **Verify Full Suite:** Re-run targeted + integration tests before finishing.

2 Safe Structural Refactoring

When splitting files or renaming shared interfaces across dozens of callers, anchor Codex to compiler and lint gates:

- **Baseline Proof:** Run `tsc --noEmit` and tests before touching code.
- **Atomic Moves:** Refactor one boundary at a time and re-typecheck after each step.
- **Inspect Cleanly:** Use `/diff` to audit every changed hunk prior to committing.

Live MCP Tool Calling Inside Your Terminal Loop

Tools wired under `[mcp_servers.*]` in `~/.codex/config.toml` become native capabilities inside Codex CLI. Instead of copy-pasting database schemas, Sentry stack traces, or GitHub issue threads into your prompt, let Codex fetch live facts directly inside the reasoning loop:

```
# 1. Start an interactive session inside workspace-write with on-request prompts
codex --sandbox workspace-write --ask-for-approval on-request

# 2. Prompt combining MCP context + TDD verification loop:
> Fetch GitHub issue #842 via the github MCP server, inspect the failing query plan
  using postgres_ro, write a regression test in tests/billing.spec.ts that fails
  against current code, fix the N+1 join, and run pnpm test billing until green.

# 3. Essential interactive slash commands during long sessions:
/diff       # Inspect colorized git diff of every edit Codex made so far
/review     # Run a dedicated second-pass bug & regression check on uncommitted changes
/compact    # Summarize older turns while preserving key file paths and decisions
/resume     # Reopen a previous terminal session or parallel Git worktree with full state
/import     # Import instructions, skills, plugins & recent chats from Claude Code, Cowork, or Cursor
```

Fast Non-Interactive Pipe

Run `codex exec "Generate changelog from last 15 commits"` inside scripts or pre-commit hooks for scripted one-shot output.

Checkpoint Before Risk

Commit or `git stash` clean work before multi-file refactors so reverting an unwanted experiment takes a single command.

Headless CI Review

Pair `codex exec --sandbox read-only` with PR diffs to catch missing null checks or migration hazards before human review.

Rapid Iteration Habit: Define “Done” as a Passing Command

Prompts ending with *“and do not finish until `pytest` and `ruff check` pass with zero warnings”* unlock `GPT-6.1 Sol`’s self-healing loop—turning first-draft syntax slips into zero-touch fixes before you look back at the terminal.

Codex Cloud Fleets: Parallel Containers to Clean PRs

When a task spans dozens of packages, takes twenty minutes of test suites, or repeats across multiple services, stop babysitting one terminal tab. **Codex Cloud** clones your repo into isolated cloud containers that run for hours or days and return review-ready PRs.

1 Connecting GitHub / GitLab & Setup

- **Repo Authorization:** Link your GitHub or GitLab organization inside Codex Cloud settings and select target repositories.
- **Environment Setup Script:** Provide a deterministic setup command (e.g., `pnpm install --frozen-lockfile`) cached into container snapshots.
- **Cached Warm Boots:** Snapshotted dependency layers launch new worker pods in seconds with your full test toolchain ready.

2 Fan-Out & PR Review Loops

- **Parallel Task Fan-Out:** Dispatch 4–10 independent tasks simultaneously—each gets its own branch and isolated container.
- `/cloud` **Terminal Handoff:** Escalate local investigations straight from Codex CLI into Cloud without losing context.
- **PR Comment Follow-Ups:** Tag Codex on an open PR or push review notes; the container resumes, fixes feedback, and updates commits.

```
# Dispatch parallel Codex Cloud attempts from CLI or open interactive TUI via: codex cloud
codex cloud exec --env env_core_api --branch main --attempts 3 "Migrate v1 auth endpoints to OAuth2 PKCE"
codex cloud status <task-id>                       # Check progress & PR links (or codex cloud list)
codex cloud diff <task-id> && codex cloud apply <task-id>  # Preview & apply a Cloud patch locally
```



Parallel Cloud Fleet Execution: Each translucent worker pod builds and tests its own isolated branch against your cached environment snapshot, converging verified diffs and passing CI logs into GitHub or GitLab pull requests.

5 Real-World Copy-Paste Recipes

Adapt these five battle-tested blueprints for your daily engineering workflow. Notice how each pairs the right sandbox tier with a concrete verification command or Cloud PR handoff.

Recipe & Surface	Sandbox & Tools	Copy-Paste Prompt / Command

1. Production Incident Triage
Codex CLI (Local)
	`workspace-write`
	`on-request`
	MCP: Sentry / Logs
		“Inspect the 500 spike on `POST /v2/checkout` using our error-logs MCP. Trace the stack frame in `src/checkout/`, write a failing unit test reproducing the null payload crash, apply the minimal guard fix, and confirm `npm test` passes.”

2. Cross-Repo SDK Migration **Codex Cloud Fleet**	Cloud Containers `--attempts 3` GitHub PR Output	“Dispatch via `codex cloud exec --env <id>`: Update deprecated `v1.Client` imports to `v2.UnifiedClient` across `packages/web`, `packages/mobile`, and `packages/cli`. Follow each package’s `AGENTS.md`, run full typechecks and tests, and open one clean PR per package.”

3. Test Harness Generation
Codex CLI (Local)
	`workspace-write`
	`on-request`
	`SKILL.md` QA module
		“Analyze `src/billing/proration.ts` for leap-year, negative credit, and timezone rollover edge cases. Generate a Vitest suite in `tests/proration.test.ts` targeting 95%+ branch coverage without mocking core math helpers. Run coverage until all cases pass.”

4. Read-Only Security Audit
Codex CLI (Safe)
	`read-only`
	`on-request`
	Zero Disk Writes
		“Launch with `codex --sandbox read-only`: Audit every route in `api/routes/` for missing tenant ownership checks (IDOR), unvalidated SQL interpolation, and SSRF URL fetches. Output a severity-ranked Markdown report with exact `file:line` citations.”

5. Overnight Cloud Backlog Batch **Codex Cloud Fleet**	Cloud Containers Async Overnight `AGENTS.md` Rules	“Queue 5 independent Codex Cloud tasks for issues tagged `good-for-codex` (#301–#305): in each isolated pod, add tests first, implement the fix strictly within the module boundaries in `AGENTS.md`, verify lint + tests green, and leave draft PRs ready for morning review.”

Recipe Scaling Pattern: Local Prototype → Cloud Fleet

Whenever you discover a multi-step fix that works cleanly on one service in Codex CLI (Recipe 1), freeze the steps into a repo `SKILL.md` file and launch Recipe 2 in Codex Cloud to roll out the exact same pattern across ten sibling services simultaneously.

Security, Secret Hygiene & Enterprise Governance

Autonomous agents move fast—which makes clear boundaries non-negotiable. Codex separates **build-time setup secrets** from **runtime environment access** and disables untrusted outbound networking by default so credentials never leak through generated code or logs.

Never Paste Production API Keys into Prompts, `AGENTS.md`, or `SKILL.md`

Prompt files and repo markdown are transmitted to model context and committed to Git. Store credentials strictly in OS environment variables, secret managers, or Codex Cloud's encrypted environment vault—never inside text instructions.

Network Egress Controls (Local & Cloud)

- **Default-Deny Outbound:** Inside `workspace-write` and Codex Cloud execution phases, outbound internet access is disabled by default so agents cannot exfiltrate source code or pull unvetted binaries mid-run.
- **Two-Phase Cloud Lifecycle:** Network access is enabled only during the pre-flight *setup phase* (to install npm/pip packages), then severed before the agent reads prompts or writes code.
- **Domain Allowlists:** Enterprise admins can permit specific internal registries or test endpoints (e.g., `registry.npmjs.org`, internal PyPI) while blocking arbitrary HTTP calls.

! Secret Injection & Scoping

- **Setup Secrets vs. Agent Secrets:** Keep private package registry tokens scoped strictly to the container setup step; strip them before agent execution starts.
- **Least-Privilege Test Keys:** Inject only ephemeral staging or local-mock tokens into Cloud test environments—never production DB or billing credentials.
- **MCP Read-Only Wrappers:** Configure `~/.codex/config.toml` MCP tools (like Postgres) with read-only database roles for local triage.

Enterprise Audit Logs & CI/CD Review Gates

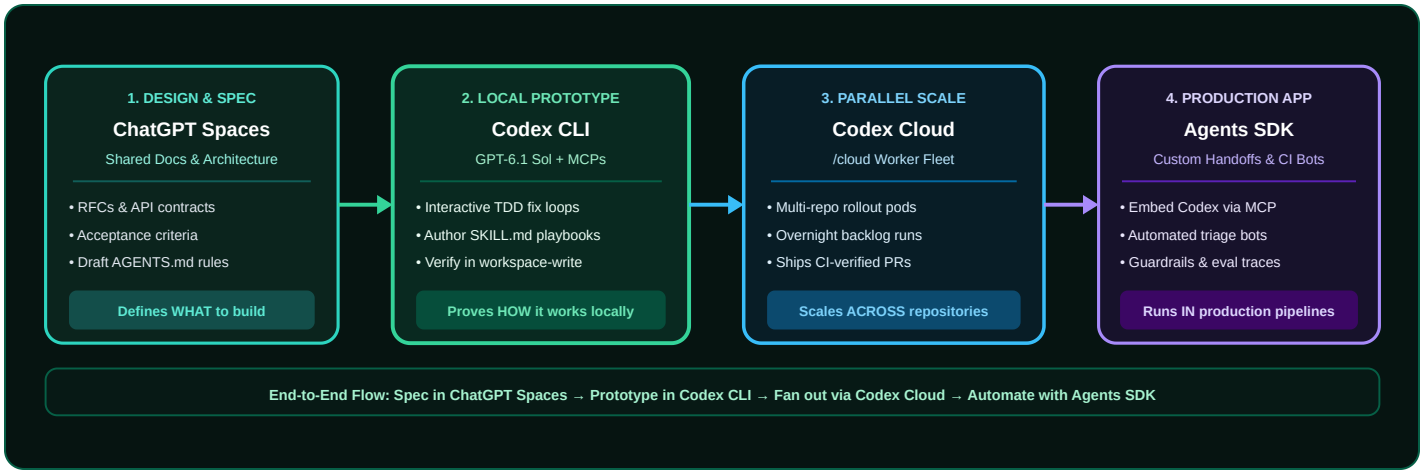
Governance Control	How Codex Enforces It	Recommended Team Policy
1. Workspace Audit Trail	Every CLI command, MCP tool call, Cloud container log, and diff artifact is logged with timestamps and user identity.	Stream Team/Enterprise audit logs to your SIEM and retain Cloud container test transcripts alongside PR links.
2. Zero Auto-Merge Rule	Codex Cloud opens pull requests or branches but leaves merge authority with GitHub/GitLab branch protection.	Require at least one human reviewer plus passing CI checks (`CODEOWNERS` + secret scanning) before merging.
3. Managed `config.toml` Baseline	Admins can enforce organization-wide sandbox floors and block `danger-full-access` escalation on corporate laptops.	Pin default `sandbox_mode = "workspace-write"` and `approval_policy = "on-request"` with restricted egress in managed profiles.

Prompt Injection Defense: Separate Untrusted Data from Write Privileges

When asking Codex to summarize external web pages, third-party GitHub issues, or untrusted logs, keep the session in `read-only` sandbox and review proposed shell tool calls under `on-request` policy so hidden instructions in external text cannot trigger unintended writes.

Cross-Tool Synergy: From Terminal Prototype to Production Fleet

Codex does not live in isolation. It forms the engineering backbone of OpenAI's broader ecosystem—connecting rapid local terminal exploration, scalable Cloud pull-request fleets, and custom production agents built with the OpenAI Agents SDK or ChatGPT Spaces.



Walkthrough: Shipping a Multi-Service Feature End-to-End

1 Spec & Local Proof (Spaces → Codex CLI)

Your team aligns on an API schema change inside **ChatGPT Spaces**, exporting the spec into your repo. You open **Codex CLI** in `workspace-write` mode, refine the core TypeScript handler with `GPT-6.1 Sol`, and capture the migration steps in a reusable `SKILL.md` module.

2 Fleet Rollout & Automation (Cloud → Agents SDK)

You trigger `/cloud` to roll the verified SDK change across four downstream repos concurrently, review the resulting GitHub PRs, and wire an **OpenAI Agents SDK** triage workflow (or `codex exec` CI check) to guard future commits automatically.

Unifying Artifact: Version-Controlled `AGENTS.md` & `SKILL.md`

Because every stage of the pipeline reads the same plain-Markdown rules committed alongside your source code, lessons learned from a human PR review feed straight back into `AGENTS.md`—making both tomorrow's CLI session and next week's Cloud fleet smarter.

Troubleshooting Clinic & Pre-Flight Checklist

Most Codex CLI or Codex Cloud hiccups trace back to one of seven predictable setup or sandbox snags. Use these ten-second fixes to get unblocked fast, then verify your repo with the pre-flight checklist.

Symptom / Error	Root Cause	10-Second Fix
1. TOML syntax or `wire_api = "chat"` error on launch	Malformed `~/.codex/config.toml` or custom provider set to legacy Chat Completions (`wire_api = "chat"`).	Validate with `python3 -c "import tomllib; tomllib.load(open('...','rb'))"` and point custom providers to a **Responses API** endpoint (Chat Completions-only is unsupported).
2. Sandbox permission denied on write or install	Session is running in `read-only` or trying to write outside the jailed workspace root.	Switch to `--sandbox workspace-write` for project edits, or install global tools manually outside Codex before retrying.
3. Custom `SKILL.md` never triggers automatically	Missing YAML frontmatter, file not named `SKILL.md`, or vague `description` field.	Confirm path is `.agents/skills/<name>/SKILL.md`, rewrite `description` with explicit trigger verbs, or invoke `$<name>` directly.
4. Codex ignores updated `AGENTS.md` rules	Long-running session cached earlier rules, or a subfolder `AGENTS.md` overrides root guidance.	Check active hierarchy in the session banner, start a fresh session or `/compact`, and keep rules crisp near the top.
5. MCP server times out at startup	Slow `npx` download, missing environment variable, or server printing logs to `stdout`.	Pre-install the MCP binary locally, verify `env` keys in `config.toml`, and route server logs to `stderr` only.
6. Cloud container network block during tests	Tests try to hit external APIs after the setup phase sealed outbound networking.	Install dependencies in the setup script, mock external HTTP calls in tests, or allowlist required test hosts in org settings.
7. Dirty worktree or Cloud PR merge conflict	Uncommitted local edits collided with `codex cloud apply`, or base branch moved mid-run.	`git stash` local changes before applying Cloud patches, or ask Codex on the PR branch to rebase onto latest `main`.

6-Point Pre-Flight Checklist Before Dispatching Autonomous Work

- ✓ **Narrowest Sandbox Active:** Confirmed `workspace-write` (or `read-only` for audits) rather than unnecessary `danger-full-access`.
- ✓ **Clean Git Working Tree:** Committed or stashed unrelated changes before letting Codex edit or apply Cloud diffs.
- ✓ **Crisp `AGENTS.md` Commands:** Verified exact test, lint, and typecheck commands sit in the top 50 lines of `AGENTS.md`.
- ✓ **Lean Progressive Skills:** Moved specialized playbooks into `.agents/skills/*/SKILL.md` with clear trigger descriptions.
- ✓ **Secrets Out of Prompts:** Confirmed API keys live only in env vars or Cloud setup secrets, never inside Markdown files.
- ✓ **Human Review Gate Armed:** Paired autonomous runs with `/diff`, `/review`, and protected PR merge checks.